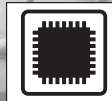
第3回

マルチプレクサと算術論理演算回路

中野浩嗣，伊藤靖朗

デバイスの記事　ビギナーズ

本連載は，さまざまな回路をVerilog HDLで設計していき，最終的には小型のCPUを実現することをねらいとしている．今回は，CPUの主要な構成部品である，算術論理演算回路を設計する．また，そのための準備運動として，3入力のマルチプレクサを設計する．（編集部）

算術論理演算回路（ALU：arithmetic logical unit）は，算術演算と論理演算を実行する回路であり，CPUを構成する主要な部品の一つです．今回は，この算術論理演算回路を設計してみたいと思います．

まず，算術論理演算回路を設計するための準備運動として，3入力のマルチプレクサを設計します．

● マルチプレクサの設計

3入力マルチプレクサは，1ビットの入力ポートa，b，cと2ビットの入力ポートf，1ビットの出力ポートsを持ちます（**図1**）．入力fは選択入力として機能します．すなわち，fが“00”のとき，出力ポートsからは，aに入力されている値が出力されます．同様に，“01”，“10”のときは，sからはそれぞれb，cの値が出力されます．また，fには“11”が入力されることはなく，もし入力されたとしても，sから出力される値は何でもよいものとします．

マルチプレクサをVerilog HDLで記述したものを**リスト1**に示します．8行目から始まるalways文で3入力マルチプレクサの動作を定めています．ここでは，a，b，c，fのいずれかの値が変化するたびに，後に続くcase～endcase間の文が実行されます．

always文の直後のcase文は，引き数（ここではf）の値によって動作を決定します．このcase文は，C言語のswitch文とよく似た機能を持っています．

10行目の2'b00は，2進表現で“00”の値を持つ2ビットの数値です（数値表現については，p.130のコラム「Verilog

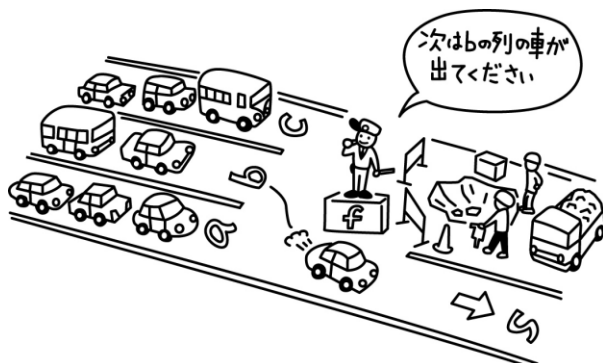



図1　マルチプレクサのイメージ

リスト1　マルチプレクサのVerilog HDL記述（multiplexer.v）

```
 1 module multiplexer(a, b, c, f, s);
 2
 3  input a, b, c;
 4  input [1:0] f;
 5  output s;
 6  reg s;
 7
 8  always @ (a or b or c or f)
 9    case (f)
10      2'b00: s = a;
11      2'b01: s = b;
12      2'b10: s = c;
13      default: s = 1'bx;
14    endcase
15
16 endmodule
```

（10～13行目）fの値によってsに代入される値が変わる

KeyWord　Verilog HDL，FPGA，算術論理演算回路，ALU，マルチプレクサ，数値表現，不定値，非同期ラッチ

HDLの数値表現」を参照)．ここでは，fの値が"00"のときに，レジスタ型変数sにaの値を代入することを表しています．11行目と12行目は，それぞれfが"01"と"10"の場合の動作を定義しています．

13行目は，fがこれらの値以外の場合の動作を定めています．ここでは，sに1'bxが代入されていますが，このxは不定値である(sに書き込まれる値はどのようなものでもかまわない)ことを意味しています．このように定義しておくと，設計ツールはfが"11"のときのsの値を考慮しなくてすむので，よりコンパクトで高速な組み合わせ回路を生成することが期待できます．

ここでは，13行目のdefault文は「省略可能ではないか?」と思われるかもしれません．しかし，省略すると，fの値が"11"のときにはsの値は変更されないことになります．つまり，sの値は「現在の値を保持し続ける」ことになり，設計ツールはそのために非同期ラッチを生成してしまいます．今回のように，always文中にcase文を用いて組み合わせ回路を設計する場合は，その最後に必ずdefaultを含むようにしましょう(非同期ラッチの生成については，p.132のコラム「always文による非同期ラッチの生成」を参照)．

図2
算術論理演算回路aluのブロック図
aとbを入力し，sを出力する．演算機能はfで選択する．

● 算術論理演算回路の設計

マルチプレクサの考え方に基づいて，算術論理演算回路を設計します．ここでは，二つの16ビットの入力ポート(aとb)，機能選択を行うための5ビットの入力ポート(f)，16ビットの出力ポート(s)を持つ算術論理演算回路を考えます(**図2**)．

算術論理演算回路の機能は自由に決めることができますが，ここでは以下のように定めます(**表1**)．機能選択ポートfのビット数を増やすことにより，より多くの演算をサポートすることも可能です．

なお，この算術論理演算回路がCPUを構成する部品となることを考慮し，入力ポートaとb，出力ポートsのビット列が数値を表す場合，そのビット列は2の補数表現であるとして演算を行うことにします[注1]．

1)算術演算

加算(ADDition)，減算(SUBtraction)，乗算(MULtiplication)，符号反転(NEGation)をサポートします．

2)ビット・シフト演算

左ビット・シフト(SHift Left)と右ビット・シフト(SHift Right)をサポートします．**表1**に示した式は，bのビット列をaの値の分だけシフトすることを表しています．

注1:補数表現については，前回の記事(本誌2007年6月号のpp.130-131)を参照のこと．

コラム1
Column Verilog HDLの数値表現

Verilog HDLの数値表現は，基数を指定することによって，2進数表現や16進数表現を指定することができます．指定しなければ10進数表現とみなされます．基数の指定は，bまたはBで2進数(binary)を，oまたはOで8進数(octal)を，dまたはDで10進数(decimal)を，hまたはHで16進数(hexadecimal)を表します．また，ビット数も指定できます．指定しなければ32ビットとして扱われます．

基数とビット数を指定する場合の形式は，

[ビット数]'[基数][数]

です．**表A**に，各数値表現と対応するビット列の具体例を示します．

なお，Verilog HDLでは，1ビットの値は'0'，'1'，'z'(ハイ・インピーダンス)，'x'(不定値)の4通りの値をとることができます．ハイ・インピーダンスについては次回以降で詳しく説明しますが，直感的には，値がない状況を表すのに用います．

不定値は，論理設計でのドントケア(don't care)を表現するのに用いられます．また，回路シミュレーションでは，一つの信号線に複数の値が同時に書き込まれた場合や値が一つも書き込まれないときなどに，その値は不定値となります．

表A Verilog HDLの数値表現とビット列の具体例

数値表現	ビット列
15	00000000 00000000 00000000 00001111
−3	11111111 11111111 11111111 11111101
7'b110	0000110
b110	00000000 00000000 00000000 00000110
16'hfffe	11111111 11111110
4'hz	zzzz
4'hx	xxxx

表1
算術論理演算回路aluの仕様
算術論理演算回路のサポートする演算は，大きく「2項演算」と「単項演算」に分けられる．2項演算では，演算結果sは入力aとbの両方に依存する．単項演算では，sは入力aのみに依存する．

サポートする演算の種類		機能選択f		出力s
2項演算	算術演算	ADD	00000	b＋a（加算）
		SUB	00001	b－a（減算）
		MUL	00010	b＊a（乗算）
	ビット・シフト演算	SHL	00011	b << a（左ビット・シフト）
		SHR	00100	b >> a（右ビット・シフト）
	ビットごとの演算	BAND	00101	b & a（ビットごとの論理積）
		BOR	00110	b | a（ビットごとの論理和）
		BXOR	00111	b ^ a（ビットごとの排他的論理和）
	論理演算	AND	01000	b && a（論理積）
		OR	01001	b || a（論理和）
	関係演算	EQ	01010	b==a（bとaは等しい）
		NE	01011	b!=a（bとaは等しくない）
		GE	01100	b>=a（bはa以上）
		LE	01101	b<=a（bはa以下）
		GT	01110	b>a（bはaより大きい）
		LT	01111	b<a（bはaより小さい）
単項演算	算術演算	NEG	10000	－a（符号反転）
	ビットごとの演算	BNOT	10010	~ a（ビットごとの反転）
	論理演算	NOT	10001	!a（論理否定）

3）ビットごとの演算

ビットごとの論理積（Bitwise AND），ビットごとの論理和（Bitwise OR），ビットごとの排他的論理積（Bitwise XOR），ビットごとの反転（Bitwise NOT）をサポートします．これらの演算では，ビット列のビットごとに独立に論理演算を行います．

4）論理演算

論理演算は，C言語と同じように，値が0の場合は偽，0でない場合は真とみなします．ここで論理積（AND）`b && a`は，aとbが両方とも「0でない」（つまり，16ビットのいずれかが'1'）場合，出力sは1（16'h0001）となります．また，aとbのいずれか一つでも「0である」（つまり，全16ビットが'0'）場合，出力sは0（16'h0000）となります．

同様に，論理和（OR）`b || a`は，aとbが両方とも「0である」場合に0となり，いずれか一つでも「0でない」場合に1となります．

論理否定（NOT）`!a`は，aが0のとき1となり，0でないとき0となります．

5）関係演算

関係演算は，aとbの大小関係を判定します．演算結果は1（真）または0（偽）となります．

表2　Verilog HDLの主な演算子

算術演算子	
`+`	加算
`-`	減算
`*`	乗算
`/`	除算
`%`	剰余算
`-`	2の補数（符号反転）
ビットごとの演算子	
`~`	1の補数（ビットごとの反転）
`&`	ビットごとの論理積
`|`	ビットごとの論理和
`^`	ビットごとの排他的論理和
`~^`	ビットごとの排他的論理和否定
ビット・シフト演算子	
`<<`	左辺を右辺だけ左シフト
`>>`	左辺を右辺だけ右シフト
論理演算子	
`!`	論理否定
`&&`	論理積
`||`	論理和

関係演算子	
`==`	左辺と右辺は等しい
`!=`	左辺と右辺は等しくない
`>=`	左辺は右辺以上
`<=`	左辺は右辺以下
`>`	左辺は右辺より大きい
`<`	左辺は右辺より小さい
リダクション演算子	
`&`	全ビットの論理積
`~&`	全ビットの論理積否定
`|`	全ビットの論理和
`~|`	全ビットの論理和否定
`^`	全ビットの排他的論理和
`~^`	全ビットの排他的論理和否定
条件演算子	
`?:`	条件?真の場合:偽の場合
連結演算子	
{,}	ビット列の連結
反復演算子	
{{}}	ビット列の繰り返し

● 算術論理演算回路の設計

算術論理演算回路は，**表2**の演算子とマルチプレクサで利用した考え方を用いて設計することができます．**表1**の演算結果をマルチプレクサ風に処理し，一つだけをsに出力します．**リスト2**は，そのVerilog HDL記述です．

1行目～19行目の`define`文で，ADDなどの機能を選択するための値を定義します．`define`文の前には「`」（バッククオート）がついていることに注意してください．

`define`文は，C言語の`#define`文と本質的に同じです．ここでの`define`文は数値を定義するだけで，回路が作られるわけではありません．34行目～52行目で用いられている`` `ADD ``などが，`define`文で定義された数値`5'b00000`などに置き換えられます．`ADD`に定義された値

リスト2　算術論理演算回路のVerilog HDL記述（alu.v）

```
 1 `define  ADD  5'b00000
 2 `define  SUB  5'b00001
 3 `define  MUL  5'b00010
 4 `define  SHL  5'b00011
 5 `define  SHR  5'b00100
 6 `define  BAND 5'b00101
 7 `define  BOR  5'b00110
 8 `define  BXOR 5'b00111
 9 `define  AND  5'b01000
10 `define  OR   5'b01001
11 `define  EQ   5'b01010
12 `define  NE   5'b01011
13 `define  GE   5'b01100
14 `define  LE   5'b01101
15 `define  GT   5'b01110
16 `define  LT   5'b01111
17 `define  NEG  5'b10000
18 `define  NOT  5'b10001
19 define  BNOT 5'b10010
20
21 module alu(a, b, f, s);
22
23   input [15:0] a, b;
24   input [4:0]        f;
25   output [15:0] s;
26   reg [15:0]          s;
27   wire [15:0] x, y;
28
29   assign x = a+16'h8000;
30   assign y = b+16'h8000;
31
32   always @(a or b or x or y or f)
33     case(f)
34       `ADD : s = b + a;
35       `SUB : s = b - a;
36       `MUL : s = b * a;
37       `SHL : s = b << a;
38       `SHR : s = b >> a;
39       `BAND: s = b & a;
40       `BOR : s = b | a;
41       `BXOR: s = b ^ a;
42       `AND : s = b && a;
43       `OR  : s = b || a;
44       `EQ  : s = b == a;
45       `NE  : s = b != a;
46       `GE  : s = x >= y;
47       `LE  : s = x <= y;
48       `GT  : s = x > y;
49       `LT  : s = x < y;
50       `NEG : s = -a;
51       `BNOT : s = ~a;
52       `NOT : s = !a;
53       default : s = 16'hxxxx;
54     endcase
55
56 endmodule
```

を用いる場合，前に「`」を付けて「`ADD」とします．このようにdefine文を用いて定義しておけば，数値の割り当ての変更があった場合にそこだけ変更すればよいので，バグが発生しにくくなります．

32行目～54行目のalways文で，組み合わせ回路としての算術論理演算回路の動作を定めています．33行目のcase文で，fの値によってsに代入する値が選択されます．fの値に該当するものがない場合（例えば5'b11111の場合），sには不定値16'hxxxxが代入されます（sの値はどのようなものでもかまわないことを意味する）．

27行目で，16ビットのネットxとyを宣言しています．29行目と30行目のassign文で，これらの値は，aとbに16'h8000を加算したものと定められています．このxとyは，46行目～49行目の関係演算の大小比較において，aとbの代わりに用いられています．これは，Verilog HDLのビット列が符号なし2進数表現として扱われることへの対策です．つまり，16'h8000を加算することにより，負の数を含む2の補数表現を，非負整数に変換しているのです（**表3**）．非負整数なので，符号なし2進数と見なすことができます．また，aとbの両方に同じ値（16'h8000）を

コラム2

Column　always文による非同期ラッチの生成

always文で組み合わせ回路を作る場合にはreg文で宣言されたレジスタに値を代入しますが，あらゆる場合について代入文の実行が保証されていないと，非同期ラッチが生成されてしまいます．

例えば，**リスト2**の算術論理演算回路で，53行目のdefault :s = 16h'xxxx;を削除してしまった場合を考えてみましょう．すると，aやbの値が変わっても，fの値が5'b11111の場合，sに値が書き込まれないことになります．よって，sには以前の値がそのまま保持されます．設計ツールは，この仕様を満たすため，sの値を保持するための非同期ラッチを**図A**のように生成します．そして，fの値が5'b11111のときは，非同期ラッチが保持している値がsに出力されることになります．

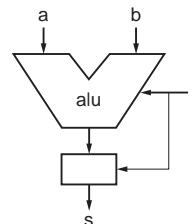
このようなクロックに関係なく値をラッチする非同期ラッチは，本連載で設計する完全同期式回路では用いることはありません．完全同期式回路では，クロックの立ち上がり時にのみ値を取り込む（ラッチする）動作を用います．

図A
意図しない非同期ラッチが付いた算術論理演算回路

表3　16ビットのビット列に対する値(10進数で表した)

ビット列	符号なし2進数表現	1の補数表現	2の補数表現
16'h0000	0	0	0
16'h0001	1	1	1
16'h0002	2	2	2
…(中略)…			
16'h7FFE	32766	32766	32766
16'h7FFF	32767	32767	32767
16'h8000	32768	−32767	−32768
16'h8001	32769	−32766	−32767
…(中略)…			
16'hFFFE	65534	−1	−2
16'hFFFF	65535	−0	−1

加算しているので，大小関係は保存されます．よって，Verilog HDLの関係演算子をそのまま用いることができ，符号なし2進数表現xとyの大小関係は，2の補数表現aとbのそれと一致します．

● 算術論理演算回路のシミュレーション

最後に，作成した算術論理演算回路が正しく動作することを，シミュレーションで確認してみましょう．

前回と同様に，テストベンチを作成します注2．**リスト3**にテストベンチの例を示します．テストベンチなので，モジュールalu_tbにポートはありません．

5行目と6行目で，入力のためのレジスタ型変数a，b，fを宣言します．7行目で出力のためのネット型変数sを宣言します．9行目では，算術論理演算回路のモジュールaluをインスタンス化しています．11行目から22行目では，ALUの入力信号a，b，fの値を設定し，算術論理演算回路の動作を確認します．例では，最初に，aに−3，bに3，fに5'b01100を代入しています．aもbも16ビットなので，実際には，aに16'b1111111111111101，bに16'b0000000000000011が代入されます．そして，100nsごとにaの値を変えています．

波形部分を右クリックして表示されるメニューから数値の表示形式を選択し，算術論理演算回路の演算内容によって表示形式を変更します．例えば，「Decimal(Signed)」に変更すると分かりやすいでしょう(**図3**)．

入力a，b，fの値をいろいろ変えてシミュレーションを行い，出力sがどのような値になるのかを確認してみてください．

注2：本誌2007年4月号のp.111，または2007年6月号のp.132を参照のこと．

リスト3　算術論理演算回路のテストベンチ(alu_tb.v)

```
 1 `timescale 1ns / 1ps
 2
 3 module alu_tb;
 4
 5   reg [15:0] a,b;
 6   reg [4:0] f;
 7   wire [15:0] s;
 8
 9   alu alu0(.a(a),.b(b),.f(f),.s(s));
10
11   initial begin
12   a = -3; b= 3; f = 5'b01100;
13   #100 a = -2;
14   #100 a = -1;
15   #100 a =  0;
16   #100 a =  1;
17   #100 a =  2;
18   #100 a =  3;
19   #100 a =  4;
20   #100 a =  5;
21   #100 a =  6;
22   end
23
24 endmodule
```

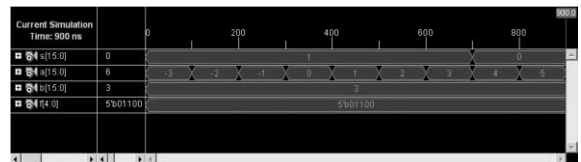



図3　算術論理演算回路のシミュレーション波形
この図より，fが5'b01100であること，つまり，関係演算GE(>=)が正しく行われていることが確認できる．

次回は，値を記憶することができるフリップフロップを用いた順序回路の設計方法について解説します．カウンタを設計し，FPGAボードを用いて動作確認を行います．

なかの・こうじ
いとう・やすあき
広島大学大学院 工学研究科

<筆者プロフィール>
中野浩嗣．1992年，大阪大学大学院 博士後期課程修了．工学博士．一つの民間企業，二つの大学を経て，2003年より広島大学 教授．
伊藤靖朗．2003年，北陸先端科学技術大学院大学 博士前期課程修了．現在，広島大学 助教授．